6.829 Computer Networks
Problem set 2

Shalev Ben-David, Yonatan Belinkov

March 25, 2013

1 Introduction

This is the project report for problem set 2 of 6.829. It details our attempts at providing a protocol for congestion control that minimizes delay while maximizing throughput. For answers to the questions on the problem set, see section 2. For discussion of our early strategies, see section 3. For details regarding our final submission, see section 4.

2 Measurements for Questions 8, 9, and 10

Figure 1 shows the performance of a protocol with a fixed window size. The best score we were able to achieve was -4.5 log(Throughput/Delay) with a window size of 15. However, the measurements were not very stable and varied by as much as 0.1 between different runs with the same window size.

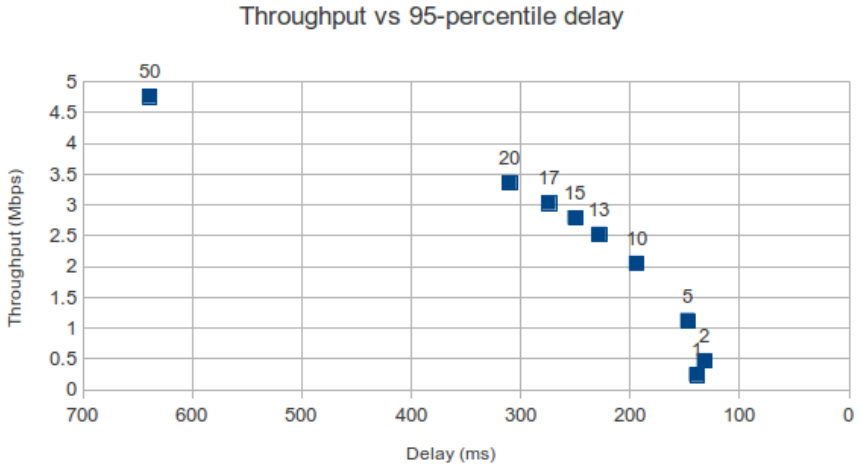

Figure 1: Throughput vs 95-percentile delay with a fixed window size

Our first try to implement an AIMD scheme did not produce very good results; we got -5.84 log(Throughput/Delay) when adding $1/w$ to the window

size on every ACK and dividing by 2 on every timeout (timeout set to 1000 ms). A slower increase of $1/w^2$ improved the score to -4.72. Another approach we tried was to change the timeout, where we found out that decreasing the timeout to 100 ms improves the score to -5.28 (while maintaining the standard AIMD). By combining both approaches we managed to improve the score up to -4.12, by using a timeout of 100 ms, an additive increase of $1/w^2$ on every ACK, and a harsher decrease of $w \leftarrow \sqrt{w}$ on every timeout.

The delay-triggered scheme proved to be competitive with AIMD. We experimented with changing the window size based on when the RTT crosses a given threshold. Table 1 summarizes the results. The best score of -4.16 was achieved with a threshold of 100 ms, an increase of 0.1,[1] and a decrease of 1.

Threshold (ms)	Increase	Decrease	Delay (ms)	Throughput (Mbps)	Score
100	1	1	308	3.39	-4.51
200	1	1	580	4.02	-4.97
50	1	1	162	1.75	-4.53
100	2	2	436	3.76	-4.75
100	1	2	299	3.12	-4.56
100	0.1	1	155	2.41	-4.16

Table 1: Delay, throughput, and log(Throughput/Delay) score with different delay-triggered schemes.

3 Early Contest Strategies

One approach we tried was to change the window size based on changes in the delivery time, i.e. the one-way trip time from the sender to the receiver. The assumption was that since we are only measured on this direction, we shouldn't include delays in receiving ACKs in the calculation. A first attempt was to compare the delivery time of the current acked packet to that of the previous packet. If the difference in delivery times was significant, we adapt the window size. Specifically, if the current delivery time is less than half of the previous delivery time, we increase the window size by one. If the current delivery time is more than two times the previous time, we decrease it by one; otherwise we do nothing. This scheme achieved a score of -4.53 (165 ms delay, 1.77 Mbps throughput).

Next we tried to reduce the frequency of changes, such that we compare to a previous delivery time only after a certain period of time has passed. Doing this every 100 ms reduced the delay (to 141 ms) but also drastically hurt the throughout (0.54 Mbps), and achieved a score of -5.74. We therefore decided to increase the window size more often, and combined our approach with an AIMD scheme, such that we increase the window size on every ACK, decrease on every timeout, [2] but also adapt it based on comparing the current delivery

[1] A non-integer increase effectively means that the window size changes only when it reaches the following integer

[2] The timeout was set to 100 ms based on the results from the previous section. A sanity check showed that a 1000 ms timeout produces much worse results.

time with a previous one. We managed to achieve a score of -4.14 (167 ms delay, 2.65 Mbps throughput) with the following parameters:

1. On every ACK, $w \leftarrow w + 1/w^2$.
2. On every timeout, $w \leftarrow \sqrt[4]{w}$.
3. Every 200 ms, compare the current delivery time to a 200-ms-ago delivery time. If $\mathsf{cur} < 0.5 * \mathsf{prev}$, $w \leftarrow w - 1$. If $\mathsf{cur} > 2 * \mathsf{prev}$, $w \leftarrow w + 1$.

Another attempt we made was based on the assumption that delivery times of individual packets may fluctuate, and we don't want to make changes based on inconsistent fluctuations. We therefore measured the average delivery time of the last k packets, and compared it with the average delivery time of the k packets that preceded them. The change in the window size was then made in a similar manner: if the current average delivery time was larger than the previous, we increment the window size; otherwise we decrement it. With this scheme we achieved a very low delay (107 ms) but at the cost of a low throughput (1.16 Mbps), with a combined score of -4.52.

4 Final Submission

The strategy in the final submission relied on measuring the rate at which acknowledgements were received, and setting the window size to be proportional to that rate. The exact formula used in the final submission was

$$w \leftarrow r * rtt_{min} * 2 + 1$$

where r is the rate at which acknowledgements were received, measured over the last 150 milliseconds. On the Verizon 140 second trace, this protocol had delay 182 and throughput 4.71, for a score of -3.65. On the full AT&T trace, this protocol had delay 354 and throughput 4.41, for a score of -4.39.

4.1 Measuring Rate and RTT

The rate of acknowledgements is measured by maintaining a queue filled with the receive times of the acknowledgements. Every new acknowledgement w=is pushed onto this queue. In addition, whenever the window size is queried, all acknowledgements older than 150 milliseconds are popped from the queue, and the size of the queue, divided by 0.15, gives the rate of acknowledgements over the last 150 milliseconds.

The minimum round-trip time was measured simply by subtracting the send time from the time the acknowledgement was received for every acknowledgement, and keeping track of the minimum value that of this figure that was observed.

4.2 Parameter Choices

The main parameters we tried to change were the scaling factor 2 and the additive constant 1. We tried finding the values of these parameters, rounded to the nearest 0.1, that give the best score. In general, decreasing either of these

parameters decreases both the throughput and the delay, so these parameters can be varied depending on the implementation needs.

Changing these factors to 1.5 and 0.5 respectively produced the best score on the Verizon 140 second trace, with a delay of 126, a throughput of 4.07, and a score of -3.43. Changing either of these parameters up or down by 0.1 decreased the score. Notice that this score is actually better than the score of -3.65 produced by the final submission. Also notice that both the delay and throughput are lower using these parameters.

The reason we did not use these parameters is that they performed significantly worse on the AT&T trace, giving a score of about -4.8. This is most likely due to over fitting as a result of the extensive parameter tuning on the Verizon data. To counter this, we selected the parameters 2 and 1, which worked reasonably well on both datasets and have the advantage of being simple round numbers.

One final thing to note is that our traces always had a roughly 1 second delay at the beginning. Since the first two seconds will not count in the final contest, we expect our protocol to produce a better score in the contest than in our tests.